WML 1.3 Language Reference

Openwave Systems Inc.
1400 Seaport Boulevard
Redwood City, CA 94063 USA
http://www.openwave.com

Part Number WMRF-13-002
October 2001

LEGAL NOTICE

Copyright © 1999–2001 Openwave Systems Inc. All rights reserved. The contents of this document constitute valuable proprietary and confidential property of Openwave Systems Inc. and are provided subject to specific obligations of confidentiality set forth in one or more binding legal agreements. Any use of this material is limited strictly to the uses specifically authorized in the applicable license agreement(s) pursuant to which such material has been furnished. Any use or disclosure of all or any part of this material not specifically authorized in writing by Openwave Systems Inc. is strictly prohibited.

Openwave, the Openwave logo, Phone.com, the Phone.com Logo, and the family of terms carrying the "UP." prefix are trademarks of Openwave Systems Inc.

All other company, brand, and product names are referenced for identification purposes only and may be trademarks that are the sole property of their respective owners.

Contents

About This Book vii
Openwave SDK vii
Audience viii
Style and Typographical Conventions viii
Code Examples ix
Related Documentation ix
WML ix
WMLScript ix
Openwave SDK ix
HDML ix
Advanced Wireless Features x
Other Documentation x
Technical Support and Other Resources x

1 Introduction 1
The Openwave Mobile Access Gateway Platform 1
The Openwave SDK 2
Overview of WML 3
WML Devices 3
WML Syntax Rules 4

2 Elements and Attributes 9
New Features Added Between WML 1.1 and WML 1.3 9
Elements 10
<a> 10
<access> 11
<anchor> 12
<b> 14
<big> 15
<card> 16
<catch> 20
<do> 22
 25
<exit> 25
<fieldset> 26

<go> 27
<head> 29
<<hr/>> 30
<i> 31
<img> 32
<input> 35
<link> 38
<meta> 41
<noop> 43
<onevent> 44
<optgroup> 45
<option> 46
<p> 48
<postfield> 48
<pre> 49
<prev> 50
<receive> 52
<refresh> 53
<reset> 54
<select> 55
<send> 58
<setvar> 60
<small> 60
<spawn> 61
 64
<table> 65
<td> 66
<tr> 66
<template> 67
<throw> 68
<timer> 69
<u> 70
<wml> 70

3 WML Quick Reference 71

Index 79

About This Book

This book provides reference information for WML 1.3 elements and attributes for use in developing WML services and applications for mobile browsers accessing an Openwave Mobile Access Gateway. This book also documents Openwave extensions to the WML language.

Wireless Markup Language (WML) is an XML-based solution developed by the WAP Forum. It was designed to address the need for a standard markup language for small, narrowband, mobile devices. Browsers used in these mobile devices are WAP compliant if they can handle WML documents.

The following table shows which Openwave Mobile Browsers support which WML versions.

WML 1.3	Mobile Browser, WAP Edition 5.0 or later
WML 1.1	Mobile Browser 4.1 Mobile Browser 3.2
WML 1.0	Mobile Browser 4.0

IMPORTANT For a complete and current list of mobile browser devices, Openwave Mobile Browser releases, and Openwave Mobile Access Gateway releases, check the Openwave Developer web site at:

`http://developer.openwave.com`.

Openwave SDK

The Openwave Software Development Kit (SDK) is a useful tool for developing, debugging, and maintaining your WML programs. Using the SDK, you can test your code from your local disk, through your own server, or through a Mobile Access Gateway. The tools in the SDK include an editor, an output window that lists transaction information, an HTTP window that displays source code, and the ability to view history, cookies, and variables.

In order to use the SDK, you must make certain that you download and install the correct release. SDK releases support the following browsers.

SDK WAP Edition 5.0	Mobile Browser, WAP Edition 5.0 or later
SDK Release 4.1	Mobile Browser 4.1 or later
SDK Release 4.0	Mobile Browser 4.0 or later
SDK Release 3.2 for WML1.1	Mobile Browser 3.1 or later

Audience

This book is intended for developers who are creating wireless WML services for mobile browser devices that are accessing an Openwave Mobile Access Gateway.

To use this book profitably, you should be familiar with the following:

- HTML, because WML has roots in HTML and is similar in syntax
- XML, because WML is an XML-based language
- Dynamic content creation using Common Gateway Interface (CGI) protocol or active server pages (ASP)
- Programming languages such as C and Perl

Style and Typographical Conventions

This manual uses different fonts to represent the information you must enter:

- `Text that appears like this` identifies command names, path names, URLs, and specific text that you must enter.
- *`Text that appears like this`* identifies placeholders or variables that you should replace with values appropriate to your environment.
- **`Text that appears like this`** identifies default WML attribute values.

Code Examples

Omitted code is indicated with ellipses. For instance, the ellipses in the following example indicate that additional code exists in this WML card definition:

```
1 <wml>
2     <card>
3       <p>
4         Your card definition goes here.
5       </p>
6         ...
7     </card>
8 </wml>
```

Line numbers that appear in some code examples are for reference only and are not part of the actual code.

Related Documentation

WML

The *WML 1.3 Developer's Guide* is a companion book to this book. It provides detailed instructions for implementing WML services, including the use of advanced features.

WMLScript

WMLScript compliments WML by providing code that runs within the WML document while it is displayed on the mobile browser device.

The *WMLScript 1.2 Developer's Guide* provides detailed instructions for implementing WMLScript programs.

The *WMLScript 1.2 Language Reference* provides detailed information about the WMLScript programming language.

Openwave SDK

The SDK is a useful environment for developing and debugging WML code.

The *Getting Started Guide* provides instructions for installing and getting started with the Openwave SDK and the basics of creating a wireless service. It also describes the operation of the browser user interface in detail.

HDML

HDML was the predecessor to WML and is supported by translation on the Mobile Access Gateway 5.0.

The *HDML 3.0 Reference* provides reference information on each HDML statement and option.

The *HDML Developer's Guide* provides instructions for developers for implementing a HDML service.

Advanced Wireless Features

The *Tools and API Reference* provides reference information about the tools and APIs available with Openwave SDKs. These tools consist of Perl and C libraries that automate some of the processes associated with creating a wireless service.

The *Push Library Developer's Guide* provides detailed reference information and programming information for developing services that push information and services to mobile browser device users.

Other Documentation

For a complete list of available documentation, see the Openwave Developer site at:

`http://developer.openwave.com`

Technical Support and Other Resources

The best resource for up-to-date information on developing wireless web services is the Openwave Developer site at:

`http://developer.openwave.com`

You can download tools and find a variety of useful resources, including Frequently Asked Questions, bug reporting, technical support, and an interactive developer forum.

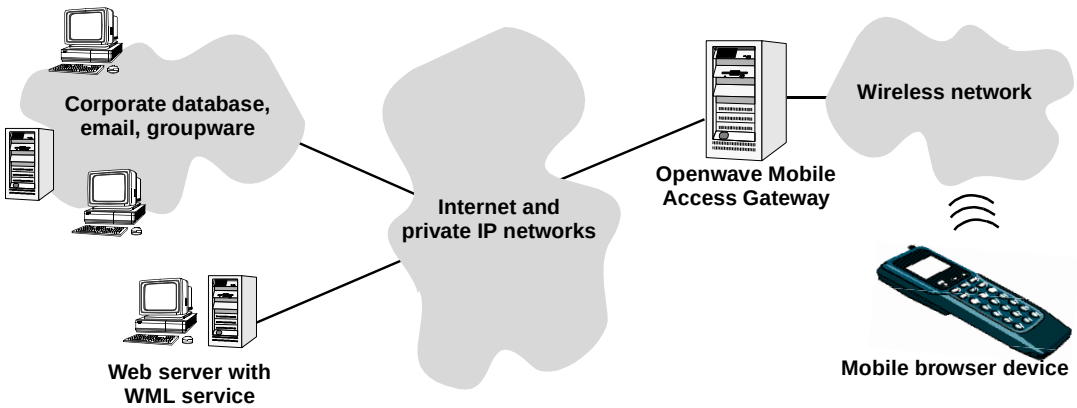
1 Introduction

This chapter introduces Wireless Markup Language (WML) and provides a high-level description of its structure and syntax. For a complete description of WML elements and attributes, see Chapter 2.

The Openwave Mobile Access Gateway Platform

The Openwave Mobile Access Gateway platform, shown in Figure 1-1, provides subscribers with secure wireless access to a wide array of Internet and other network services using detachable wireless phones or personal digital assistants (PDAs). In this documentation, these specially enabled handheld devices are called *mobile browser devices*.

Figure 1-1. The Openwave Mobile Access Gateway platform



Using a mobile browser device is much like using a conventional web browser. The user presses keys to navigate and to request URLs. Unlike standard browsers, which use Hypertext Markup Language (HTML) to display information on a computer screen, mobile browser devices use Wireless Markup Language (WML), an open language developed by the Wireless Application Protocol (WAP) Forum[1] to accommodate small

1. The WAP Forum is a wireless industry organization dedicated to promoting open wireless communication standards. For information on WAP, including specifications for WAP protocols, see http://www.wapforum.org.

handheld devices. Like HTML, WML is tag-based and supports text and image presentation, data input, and forms.

The mobile browser device uses the data capabilities of conventional wireless networks to send user requests to the Openwave Mobile Access Gateway Server,which converts these requests into Hypertext Transport Protocol (HTTP) requests and forwards them over the Internet. When the target service responds, the Openwave Mobile Access Gateway Server relays that information back to the mobile browser device.

The Openwave Mobile Access Gateway Server is the core of the Openwave Mobile Access Gateway platform. Its ability to serve as an HTTP proxy for mobile browser devices lets subscribers access any site on the World Wide Web. Some information providers also offer *WML services*, which use WML to take advantage of the mobile browser device interface.

In addition to HTML translation, the Openwave Mobile Access Gateway Server provides a number of other services. It acts as an information gatekeeper by maintaining a database of mobile browser devices and their access privileges. It also provides configuration parameters that allow administrators to monitor or log transactions.

The Openwave SDK

The Openwave Software Development Kit provides tools for developers who want to create or maintain WML services. The Openwave SDK consists of the following:

- The Openwave Browser, a Windows 95/98/NT/2000/ME application that simulates the behavior of a mobile browser device and makes it easy to test WML services
- Tools for requesting and installing SSL (Secure Sockets Layer) certificates
- Sample WML and WMLScript files and application source code, which you can use as a starting point for your own WML services
- Developer documentation consisting of online manuals in HTML format. Adobe Acrobat Portable Document Format (PDF) versions are available from the Openwave Developer site, `http://developer.openwave.com`.

  For a description of the different Openwave SDK manuals, see “Related Documentation” in the “About This Book.”

Overview of WML

WML is a markup language based on Extensible Markup Language (XML). It is designed for specifying user interface behavior and displaying content on wireless devices such as phones, pagers, and personal digital assistants (PDAs).

The Wireless Application Protocol (WAP) Forum, an industry organization dedicated to developing open standards for wireless communication, has provided a formal specification for WML. This specification, which includes the Document Type Definition (DTD) for WML, is available on the WAP Forum Web site at:

`http://www.wapforum.org`

The document you are reading now, the *WML 1.3 Language Reference*, is based on the WAP Forum WML 1.3 specification.

For a formal XML specification, see "Extensible Markup Language (XML), W3C Proposed Recommendation," available from the W3C site:

`http://www.w3.org`

WML Devices

WML is designed to support a range of devices, which typically have the following characteristics:

- Small display size (relative to conventional personal computers)
- Limited memory and CPU size
- Low bandwidth, high-latency wireless connectivity

The devices that WML currently supports fall into two main categories:

- Phones, which typically feature text displays of 4 to 10 lines and support user input through numeric and function keys.
- Personal digital assistants (PDAs), which typically feature display resolutions of 100 X 100 pixels (or better) and support enhanced user input through keypads, pointers, or handwriting recognition.

It is expected that as handheld devices with sophisticated capabilities such as voice recognition become available, WML will also support many of them.

Because WML supports a variety of devices with different capabilities, this document describes it with reference to a "least common denominator device" or "reference device." The reference device has the following characteristics:

- A display area with a width of 12 (fixed-size) characters and a height of 4 lines, *including* one line reserved for function key labels
- Support for the ASCII printable character set
- Numeric and alphabetic character entry
- Choice selection (using arrow or numeric keys)
- Two programmable function keys, referred to as `ACCEPT` and `OPTIONS`, the labels for which appear above the key in the phone display area
- A `PREV` key for navigating backward
- Vertical scrolling with arrow keys
- Horizontal scrolling of nonwrapping lines

WML Syntax Rules

This section is a brief overview of the syntax rules for writing WML code. For examples and additional discussion, see Chapter 1 of the *WML 1.3 Developer's Guide.*

Character Set

WML uses the XML document character set—currently the Universal Character Set of ISO/IEC-10646 (Unicode 2.0)—and supports any proper subset of the Unicode character set (for example, US-ASCII, ISO-8859-1, or UTF-8). You do not need to use full Unicode (UCS-4) encoding unless you are using a character set other than UTF-8 or UTF-16.

Case

Unlike HTML, WML is case sensitive. You must specify elements, attributes, and enumerated attribute values in lowercase characters. You should also keep case sensitivity in mind when you name cards or variables. For example, `variable1`, `Variable1`, and `vaRiable1` are all different variables.

White Space

In WML, *white space* is defined as any of the following:

Character	8-bit decimal value
Newline	`10`
Carriage return	`13`
Space	`32`
Tab	`9`

Except where specifically noted, WML converts one or more contiguous newlines, carriage returns, tabs, or spaces to a single space—in other words, it ignores white space beyond a single character. For example, mobile browser devices interpret the following code samples as equivalent.

Sample 1	Sample 2
`<wml><card><p> Some text </p> </card> </wml>`	`<wml> <card> <p> Some text </p> </card> </wml>`

The examples in this manual use newlines and tabs for readability. This formatting, however, is not required (and is, in fact, removed by the Openwave Mobile Access Gateway Server before transmission to the mobile browser device).

Document Prolog

All WML decks must specify the following XML document type declaration at the beginning of each file:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN"
    "http://www.wapforum.org/DTD/wml13.dtd">
```

To add this prolog to any deck automatically, you can use the `OutputDeck()` function, one of several Perl utilities provided in the Openwave SDK that help automate routine aspects of creating a WML service. For information about the `OutputDeck()` function, see the *Tools and API Reference*.

NOTE If you use the Openwave extensions in your WML content, you must include the following `DOCTYPE` header in the WML deck:

```
<!DOCTYPE wml PUBLIC "-//OPENWAVE.COM//DTD WML 1.3//EN"
    "http://www.openwave.COM/dtd/wml13.dtd" >
```

IMPORTANT Applications can use these extensions when they are accessed by a phone running Openwave Mobile Browser 4.0 or later. To identify the WAP browser type and version, capture the WAP-standard `USER_AGENT` header in the HTTP request. See the `whoami.cgi` example for implementation details.

Content Type

To create a valid WML message entity (MIME type), you must specify the following content type header before the document prolog:

```
Content-type: text/vnd.wap.wml
```

The Openwave SDKs include utilities that automate the process of creating valid HTTP responses. For more information, see the *WML 1.3 Developer's Guide* and the *Tools and API Reference*.

IMPORTANT You must leave at least one blank line between the content type and the document prolog. If you omit this line, the compiler generates an error.

Attributes

Most WML elements have one or more *attributes*, some of which are required and some of which are optional. Attributes let you specify additional information about how the device should handle the element. Although the exact syntax of a WML statement depends on whether or not the element has content, attributes always appear in the element start tag:

```
<element a1="value1" a2="value2" ...> content </element>

<element a1="value1" a2="value2" .../>
```

You must enclose attribute values in single (') or double (") quotation marks and separate each attribute-value pair with white space (as described earlier in this section). White space is *not* allowed, however, between the attribute name, equal sign, and attribute value.

Referencing Variables

Much like UNIX shell variables, you can use variables in your WML code in formatted text, URLs, selection items, or default values. To substitute a variable value, reference the variable using the syntax `$(myvar)`

The Openwave Mobile Browser automatically applies URL escaping rules based on the context. However, if you want to override the normal escaping rules for variables within URL strings, you can use the following options to force a particular behavior.

Variable reference	Description
`$(myvar:escape)`	Forces *escaping* of symbolic characters
`$(myvar:noescape)`	Forces *no escaping* of symbolic characters
`$(myvar:unescape)`	Forces *removal* of symbolic character escaping

The Openwave Mobile Browser ignores these options if you specify them for variables in contexts other than URL strings.

Special Characters

WML reserves the <, >, ', ", and & characters. To display one of these characters in formatted text, you must specify one of the following character elements.

Character	Character element
<	`<` (less than)
>	`>` (greater than)
'	`'` (apostrophe)
"	`"` (quotation marks)
&	`&` (ampersand)
$	`$$` (dollar sign)
Non breaking space	` `
Soft hyphen	``

The semicolon (`;`) is part of the character element and must be included. If you omit it, the WML compiler generates an error message.

IMPORTANT Unlike HTML, you *must* use the special character element `&` to specify the ampersand character when you use CGI arguments in URL strings.

2 Elements and Attributes

This chapter provides reference information for WML 1.3 elements and attributes, and for Openwave extensions to WML. The elements are listed in alphabetical order, with available attributes for each required attribute shown in bold.

NOTE The Openwave extensions to WML are indicated by .

New Features Added Between WML 1.1 and WML 1.3

Several new features have been introduced to *WML 1.3 Language Reference* since Version 1.1.

New Attributes

- `button` attribute added to `<do>` element.
- `list` attribute added to `<select>` element.
- `popup` attribute added to `<select>` element.
- `spin` attribute added to `<select>` element.
- `radio` attribute added to `<select>` element.
- `size` attribute added to `<input>` element.
- `enctype` attribute added to `<go>` element.

New Elements

- `<hr>`
- `<pre>`

Elements

<a>

This is the short syntax form for anchors. It uses the <a> tag instead of the <anchor> tag, and can only be used to define (implied) <go> tasks that require a URL specification.

Syntax

```
<a href="url" <!-- required --> title="label">
...any valid combination of <text>, <br/> and <img> elements
</a>
```

Attributes

`title`	A label that identifies the link. If you do not specify the `title` attribute, the device uses the word “Link” as the default label. Devices use this attribute in a variety of ways. For example, they may use it to display a tool tip or to issue a voice prompt when the user selects the link. The Openwave Mobile Browser, WAP Edition, 5.0 uses the `title` as the `accept` key label when the user selects the link. To ensure compatibility on a wide range of devices, *`label`* should be a maximum of five characters.
`accesskey`	A number (0-9) that appears on the left side of the screen next to the link. If the user presses the corresponding key on the phone keypad, the phone executes the task defined by the link. We recommend that you number the links in the order in which they appear.

<access>

The <access> element specifies access control information for a WML deck. You must specify this element within the deck header along with any meta information for the deck (for more information, see "<head>" on page 29 and "<meta>" on page 41). Each deck can have only *one* <access> element. All WML decks are public by default.

Syntax

```
<head>
<access domain="domain" path="path"/>
...
</head>
```

Attributes

domain	The URL domain of other decks that can access cards in the deck. The default value is the domain of the current deck.
path	The URL root of other decks that can access cards in the deck. The default value is "/" (the root path of the current deck) which lets any deck within the specified domain access this deck.

NOTE For a detailed explanation and examples of how the domain and path attributes work together, see the *WML 1.3 Developer's Guide*.

<anchor>

The <anchor> element anchors a task to a string of formatted text, often called a *link*. You can specify a link in any formatted text or image. When a user selects the link and presses ACCEPT, the device executes the task.

Syntax

```
<anchor title="label">task text</anchor>
```

where *task* represents the action to perform when the user activates the link and *text* is the text the device will display to represent the link:

task	You must anchor *one* of the following task elements to a link: • <go> (see "<go>" on page 27) • <prev> (see "<prev>" on page 50) • <refresh> (see "<refresh>" on page 53) • <spawn> (see "<spawn>" on page 61 • <exit> (see "<exit>" on page 25) • <throw> (see "<throw>" on page 68)
text	Devices typically set this text off from surrounding text, for example, by enclosing it in square brackets (see example) or underlining it if the device can display bitmap images.

Attributes

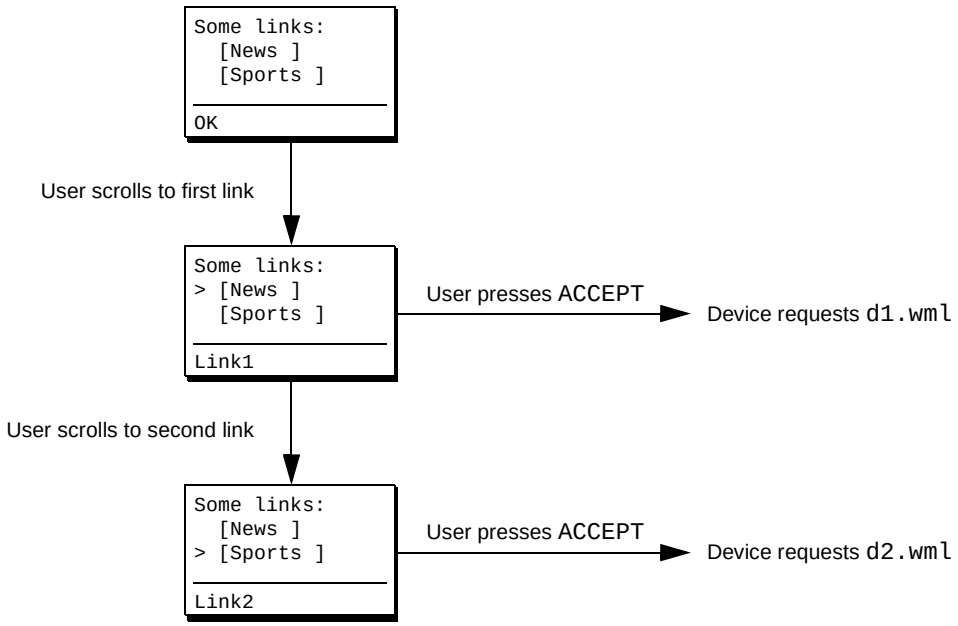
title	A label that identifies the link. If you do not specify the title attribute, the device uses the word "Link" as the default label. Devices use this attribute in a variety of ways, such as to display a tool tip or issue a voice prompt when the user selects the link. The Mobile Browser uses the title as the ACCEPT key label when the user selects the link. To ensure compatibility on a wide range of devices, *label* should be a maximum of five characters.
accesskey	A number (0–9) that appears on the left side of the screen next to the link. When the user presses the corresponding key on the phone keypad, the phone executes the task defined by the link. You should number the links in the order in which they appear.

Listing 2-1. `<anchor>` **element**

The following WML defines the card shown in Figure 2-1.

```
<wml>
<card>
<p>
Some links:<br/>
<anchor title="Link1"><go href="d1.wml"/>News</anchor><br/>
<anchor title="Link2"><go href="d2.wml"/>Sports</anchor>
</p>
</card>
</wml>
```

Figure 2-1. Card containing links



<b>

The <b> element specifies bold text.

Syntax

```
<b>text</b>
```

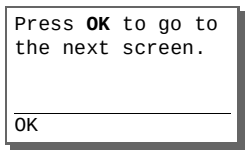
where *text* is the text to display in bold font.

Listing 2-2. <b>element

The following WML defines the card shown in Figure 2-2.

```
<wml>
<card>
<p>
Press <b>OK</b> to go to the next screen.
</p>
</card>
</wml>
```

Figure 2-2. Card containing bold text



<big>

The <big> element specifies large font text.

Syntax

```
<big>text</big>
```

where *text* is the text to display in large font.

IMPORTANT Support for this element is available when using Openwave 4.0 Browser or later.

The <br/> element specifies a line break. For example, it causes the device to display the subsequent text or image on a new line.

IMPORTANT When you define an <input> or <select> element, the Mobile Browser displays the text you specify before the element definition as a user prompt for the item. If you define *multiple* <input> or <select> elements within a card, you cannot insert line breaks within that text—if you specify the <br> element in the text flow preceding these elements, the Mobile Browser uses it as a break point for partitioning your fields into multiple screens. The net result is that only the last portion of your user prompt appears on the same screen as the entry field or selection list.

Syntax

```
<br/>
```

NOTE Unlike HTML, all WML elements require the end-element character (/). Thus, to insert a line break in WML, you must specify <br/> rather than <br>.

<card>

A WML deck consists of one or more <card> elements, each of which specifies a single interaction between the user and the device. Devices display a maximum of one card at a time—in some cases, however, a single card may appear as a series of screens. For information on related elements, see "<template>" on page 67 and "<wml>" on page 70.

Syntax

```
<wml>
<card id="name" title="label" newcontext="boolean"
ordered="true" onenterforward="url" onenterbackward="url"
ontimer="url">
    content
</card>
</wml>
```

where *content* represents the WML card definition and consists of one or more of the following elements:

content	You can specify any of the following elements in your card definition: • <onevent> (see "<onevent>" on page 44) • <timer> (see "<timer>" on page 69) • <do> (see "<do>" on page 22) • <a> (see "<a>.") • <fieldset> (see "<fieldset>" on page 26) • <img> (see "<img>" on page 32) • <input> (see "<input>" on page 35) • <select> (see "<select>" on page 55) • <p> (see "<p>" on page 48)

IMPORTANT The contents of a <card> element must be in the following order:

- <onevent>
- <timer>
- <do>

With the exception of the elements listed above, devices display these elements in the order in which you specify them.

Attributes

`id`	Specifies a name for the card. The name acts as a fragment anchor for navigating to that card. For example, you can specify `<go href="#cardname"/>` to navigate to the card (see "<go>" on page 27).
`title`	Specifies a brief label for the card. The Mobile Browser uses the label as the default bookmark name when the user bookmarks the card. Some devices might use it for other purposes, such as pop-up tooltips. **Note:** The browser does not display the title as part of the card content. The phone manufacturer may optionally display the title if there is sufficient screen real estate. The title attribute should be considered as a hint to the user and not as part of the card content.
`new context`	`true` | **`false`** Specifies whether the device should initialize the context whenever the user navigates to the card through a `<go>` task. Specifying `newcontext="true"` removes all context-specific variables, clears the history stack, and resets the device state to a well-known value.
`ordered`	**`true`** Specifies the organization of card content (see "Order options" below). `ordered="true"` causes the device to display content in a fixed sequence.
`onenter forward`	Specifies the URL to open if the user navigates to this card through a `<go>` task. This attribute represents an abbreviated form of the `<onevent>` element (see "<onevent>" on page 44).
`onenter backward`	Specifies the URL to open if the user navigates to this card through a `<prev>` task. This attribute represents an abbreviated form of the `<onevent>` element (see "<onevent>" on page 44).
`ontimer`	Specifies the URL to open if a specified `<timer>` element expires. This attribute represents an abbreviated form of the `<onevent>` element (see "<onevent>" on page 44).

Order options

The `ordered` attribute lets you specify how the device will display multiple content items within a card.

`True` displays the items in a linear sequence (the default value). Use this setting for short forms containing (mostly) required fields. If a device cannot display all the items on one screen, it divides them into multiple screens based on:

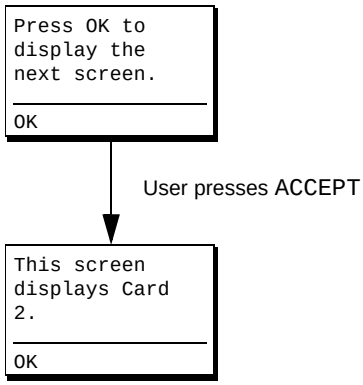
- field groupings specified by one or more `<fieldset>` elements (if any) (see "<fieldset>" on page 26).
- individual fields (with the text preceding each item definition used as a prompt)

Listing 2-3. `<card>` **element**

The following WML defines the deck shown in Figure 2-3.

```
<wml>
<card>
    <do type="accept">
        <go href="#card2"/>
    </do>
  <p>
    Press OK to display the next screen.
  </p>
</card>
<card id="card2">
 <p>
    This screen displays Card 2.
 </p>
</card>
</wml>
```

Figure 2-3. Card navigation within a deck



The two sample cards shown in Figure 2-4 illustrate the relationship between the `onenterforward` *attribute* and the `<onevent>` *element*. In this case, specifying `onenterforward` as an attribute of the `<card>` element is exactly the same as specifying the `<onevent>` element with `type="onenterforward"`—both cards produce the same results.

Figure 2-4. Equivalent methods for binding a `<go>` task to an event

```
<card onenterforward="/url">
<p>
Hello
</p>
</card>
```

```
<card>
<onevent type="onenterforward">
<go href="/url"/>
</onevent>
<p>
Hello
</p>
</card>
```

Using the abbreviated form implicitly associates a `<go>` task with the event. The main reason to use the expanded form is to associate a `<noop>`, `<prev>`, or `<refresh>` task instead (see “<onevent>” on page 44 for more information).

<catch>

The <catch> element specifies an exception handler that can process an exception passed by a throw task. Parameters sent with the exception are received with the <receive> element.

An onthrow event occurs when the exception is caught and can be bound to a task. The onthrow event can be handled with the onthrow attribute or by embedding an <onevent> inside the <catch> element.

A <spawn> element cannot contain more than one <catch> element with the same name.

Syntax

```
<catch name="error#1" onthrow="/displayError">
    <receive name="Msg"/>
</catch>
```

Attributes

name	Specifies the name of the exception. If the name attribute is missing, the <catch> element will handle any exception.
onthrow	The onthrow event occurs when an exception matches the <catch> element. Execution of the Browser continues with the URL referenced by the onthrow attribute.

Listing 2-4. <catch> **element**

The following example illustrates the use of the <catch> element in a stock trading application:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//OPENWAVE.COM//DTD WML 1.3//EN"
"http://www.openwave.com/dtd/wml13.dtd" >
<wml>
   <card id="stock">
      <do type="accept" label="Next">
         <spawn href="#order">
            <catch>
               <receive name="status"/>
            </catch>
            <receive name="status"/>
         </spawn>
      </do>
      <p>
         Stock Trade Example<br/>
         $(status)
      </p>
   </card>
   <card id="order">
      <do type="accept" label="Order">
         <spawn href="#confirm" onexit="?">
            <receive name="status"/>
         </spawn>
      </do>
      <do type="options" label="Prev">
         <exit>
            <send value="Status: No Trade"/>
         </exit>
      </do>
      <p>
         Stock Order: <br/>
         Stock  : PHCM <br/>
         Shares : 100 <br/>
      </p>
   </card>
   <card id="confirm">
      <do type="accept" label="Yes">
         <exit>
            <send value="Status: Completed"/>
         </exit>
      </do>
      <do type="options" label="Cancel">
         <throw name="onerror">
            <send value="Status: Canceled"/>
         </throw>
      </do>
      <p align="center">
         <b>*** Confirm ***<br/>
         Stock Order <br/>
         </b>
      </p>
   </card>
</wml>
```

<do>

The <do> element associates a task with an element within the user interface (for example, a function key, graphically-rendered button, or voice-activated command). When the user invokes the user interface mechanism, the device performs the associated <do> element task.

IMPORTANT The <do> element has been extended to support the specification of images as <do> element labels. This is specified with an <img> element embedded inside a <do> element. For example:

```
<do type="accept">
    <go href="/foo"/>
    <img src="img" localsrc="OK" alt="OK"/>
</do>
```

If the phone supports images, the `localsrc`, `src`, and `alt` are used, in that order. If the phone does not support images, `alt` specifies the label.

If an <img> element is specified, the label attribute on <do> is ignored. If <img> is not specified, the label attribute on the <do> element is used.

Syntax

```
<do type="type" label="label" name="name" optional="boolean">
    task
</do>
```

where *task* represents the action to perform when the user activates the specified interface mechanism:

task	You must bind *one* of the following task elements to the user interface mechanism: • <go> (see "<go>" on page 27) • <prev> (see "<prev>" on page 50) • <noop> (see "<noop>" on page 43) • <refresh> (see "<refresh>" on page 53) • <exit> (see "<exit>" on page 25) • <spawn> (see "<spawn>" on page 61) • <throw> (see "<throw>" on page 68)

Attributes

type	Required. Identifies the generic user interface mechanism that triggers the specified <do> element task (see descriptions below).
label	A label that identifies the task with the user interface mechanism. For example, if you bind a task to the ACCEPT key, the device displays this value as the function key label. If you do not specify the label attribute, the device uses the word "OK" as the default ACCEPT key label. To ensure compatibility on a wide range of devices, *label* should be a maximum of five characters. Devices ignore the label attribute if they do not support dynamic labelling. The Mobile Browser software does *not* currently support this attribute for the following TYPE values (see above): • *type="delete"* • *type="help"* • *type="prev"*
name	Specifies a name for the <do> element. If a *card-level* <do> element (for example, defined within a <card> element) has the same name as a *deck-level* <do> element (for example, defined within a <template> element), the card-level binding overrides the deck-level binding.
optional	true | **false** Specifies whether the device can ignore this element.

You can specify the following values for the `type` attribute (all types are reserved except where noted):

type `value`	**Perform task if user ...**
`accept`	Invokes `ACCEPT` mechanism (function key, button, etc.).
`delete`	Invokes `DELETE` mechanism (function key, button, etc.).
`help`	Invokes `HELP` mechanism (may be context-sensitive).
`options`	Invokes `OPTIONS` mechanism (function key, button, etc.).
`button`	Invokes `BUTTON` to be rendered on the screen.
`prev`	Navigates to card by invoking `PREV` mechanism from another card.
`reset`	Invokes `RESET` mechanism (clears or resets current device state). The Mobile Browser does *not* currently support this attribute value.
`unknown`	Invokes unknown mechanism (equivalent to `TYPE=""`). The Mobile Browser does *not* currently support this attribute value.
`vnd.`*`co-type`*	Invokes a vendor-specific mechanism where *`co`* identifies the vendor and *`type`* identifies the action (not reserved).
`X-*`, `x-*`	*Future use* (not reserved). The Mobile Browser does *not* currently support this attribute value.

None of these `type` values imply a specific user interface mechanism. Some devices map each type to a physical key, while others map them to context-dependent gestures (for example, pressing or press-holding a jog shuttle). Thus, when designing your user interface, keep in mind that you cannot specify (or assume) the particular mechanism that a device will use.

NOTE If you define multiple `<do>` elements of the same type in one card, you should specify the `name` attribute for each `<do>` element to uniquely identify each instance of the same type. When defining multiple `<do>` elements, the `OPTIONS` softkey is labeled “menu”. When the user selects the `OPTIONS` key, a popup will appear above the `OPTIONS` key. The label for each of the `<do type="options">` will be rendered in the order in which they are placed within the WML card. The options can be selected by either pressing the key accelerator associated with the item in the popup, or by using the up/down navigation, to highlight one of the options, and then pressing the `ACCEPT` key.

The element specifies emphasized text.

Syntax

```
<em>text</em>
```

where *text* is the text to display in emphasized font.

IMPORTANT Support for this element is available when using Openwave SDK, Release 4.0 or later.

<exit>

The <exit> element declares an exit task, indicating that the current context should be terminated. Values may be sent to the parent context with an embedded <send> element. The exit task causes the current context to be destroyed, including any variable and history state contained in the context.

Syntax

For example, the following will terminate the current context, and returns control to the parent context.

```
<exit>
    <send value="393"/>
    <send value="$X"/>
</exit>
```

NOTE See “<spawn>” on page 61 for example code.

<fieldset>

The <fieldset> element allows you to group multiple text or input items within a card. Specifying one or more <fieldset> elements lets you control how the device presents card content in order to simplify user navigation.

Syntax

```
<fieldset title="label">content</fieldset>
```

where *content* represents the items to group together and consists of one or more of the following elements:

content	You can specify any of the following elements in a <fieldset> group: • <fieldset> (a nested <fieldset>) • <input> (see "<input>" on page 35) • <select> (see "<select>" on page 55)
text	You can also specify formatted text in a <fieldset> group. The device uses this text in different ways depending on the elements you specify (for example, to prompt the user for input or describe various options).

Devices display these elements in the order in which you specify them.

Attributes

title	Specifies a brief label for the <fieldset> group. Some devices use the label as a title when displaying the <fieldset> content. Others might use it as a label for a user interface mechanism that lets the user navigate to the <fieldset> content. For example, if a device cannot display all card content on one screen and ordered="true" (see "Order options" on page 17), the Mobile Browser uses the title to identify this group of items on a summary-level menu.

<go>

The <go> element is a *task* element that instructs the device to open a specified URL. If the URL specifies a particular card, the device displays that card. If the URL specifies a deck, the device displays the first card in that deck.

Syntax

```
<go href="url" sendreferer="boolean" method="method"
accept-charset="charset"
<postfield name="data" value="value"/>
    content
</go>
```

where *content* represents the variables to set when opening the specified URL:

content	You can optionally specify one or more variables in a <go> statement: • <setvar> (see "<setvar>" on page 60)

IMPORTANT Unlike other WML elements that have content, specifying content for the <go> element is optional. If you do *not* specify any content, you must use the syntax <go *attributes*/> rather than <go *attributes*>*content*</go>.

Attributes

href	Required. The URL to open.
send referer	true | **false** Specifies whether the device should include the deck URL in the URL request. Specifying sendreferer="true" causes the device to set the HTTP_REFERER header to the *relative* URL of the requesting deck. If you want to restrict access to trusted services, decks that request specified URLs must set this option to TRUE.
method	**get** | post Specifies the HTTP submission method. Specifying method="post" causes the Openwave Mobile Access Gateway Server to transcode variable data to the character set specified by the HTTP headers defined in your application. You should perform this transcoding if non-ASCII characters (specifically UTF-8) may exist in the data being passed. For more information on character sets and HTTP headers, see the *WML 1.3 Developer's Guide*. If you do *not* specify the method attribute but *do* specify the postfield nested element, the device automatically uses the post method.

accept-charset	Specifies the character encodings that your application can handle. The device uses this attribute to transcode data specified by the postfield element. The Openwave Mobile Access Gateway Server assumes UTF-8 as the default encoding (of which US-ASCII is a subset), so WML services in the United States, Canada, or Australia do not need to use this attribute. You can also omit this attribute if you specify your character set(s) in the HTTP response header. Note that the accept-charset attribute overrides any character encodings you specify in the HTTP header. The syntax for this attribute is a comma- or space-delimited list of IANA character sets. For example, accept-charset="UTF-8, US-ASCII, ISO-8859-1". • For a list of supported encoding names, see the *WML 1.3 Developer's Guide*. • To view the complete IANA Character Set registry, go to http://www.iana.org/.
enctype	The attribute specifies the content type used to submit the parameter to the server (when the value of method is post). The default value for this attribute is application/x-www-form-urlencoded. Currently, only application/x-www-form-urlencoded or multipart/form-data can be specified. When the field values in a submitted form may contain characters not in the US-ASCII character set, it is recommended that the post method and multipart/form-data [RFC2388] are used. User agents must explicitly specify content type for each part. If a part corresponds to a postfield element, its content type should be text/plain. The charset parameter is required when the content contains characters not in the US-ASCII character set.

Listing 2-5. <go> **element**

The following example illustrates the syntax for the <go> element:

```
<wml>
<card title="Employee" ordered="true">
    <do type="ACCEPT" label="Find">
        <go method="post" href="?next=list">
          <postfield name="next" value="list"/>
          <postfield name="L" value="$last"/>
          <postfield name="F" value="$first"/>
         </go>
    </do>
    <p>
    Partial First Name:
    <input title="First Name" name="first"/>
    Partial Last Name:
    <input title="Last Name" name="last"/>
</card>
    </p>
</wml>
```

<head>

The <head> element specifies information about the deck as a whole, including metadata and access control information.

Syntax

```
<head>
    content
</head>
```

where *content* represents deck-level header information:

content	You can optionally specify either of the following elements in a WML deck header: • <access>—*one* only (see "<access>" on page 11) • <meta>—one or more (see "<meta>" on page 41)

Listing 2-6. <head> **element**

The following example illustrates how to specify the maximum age of a deck. The WML definition includes a <meta> statement that uses cache-control within the <head> element:

```
<wml>
<head>
    <meta http-equiv="Cache-Control" content="max-age=3600"
    forua=”true”/>
</head>
<card>
    ...
    ...
</card>
<card>
    ...
    ...
</card>
</wml>
```

<<hr/>>

The default behavior of the <hr/> element will cause a horizontal rule to be drawn across the full width of the device display. The <hr> element accepts two attributes, size and width.

If the size (thickness) of the line exceeds the dimensions of the screen, it will scroll off the bottom of the display.

Syntax

Example:

```
<hr size="3" width="75%"/>
```

Attributes

size	The size will control the height of the line to be drawn in pixels, and has an implied value of 1. Acceptable values are integers.
width	The width value controls how many pixels long (horizontally) the line should be drawn. Values can be either an absolute number of pixels, or a percentage of the screen width. If the value of the width attribute exceeds the dimensions of the screen, the line will be truncated at the screen's edge.

<i>

The <i> element specifies italic text.

Syntax

```
<i>text</i>
```

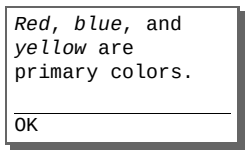
where *text* is the text to display in italic font.

Listing 2-7. <i> element

The following WML defines the card shown in Figure 2-5.

```
<wml>
<card>
<p>
<i>Red</i>, <i>blue</i>, and <i>yellow</i> are primary colors.
</p>
</card>
</wml>
```

Figure 2-5. Card containing italic text



The <img> element instructs the device to display an image within formatted text. Note that not all devices can display images.

NOTE The <do> and <option> elements have been extended by Openwave to support the specification of images—a feature unavailable in WAP-only WML. This is specified with an <img> element embedded inside a <do> or <option> element.

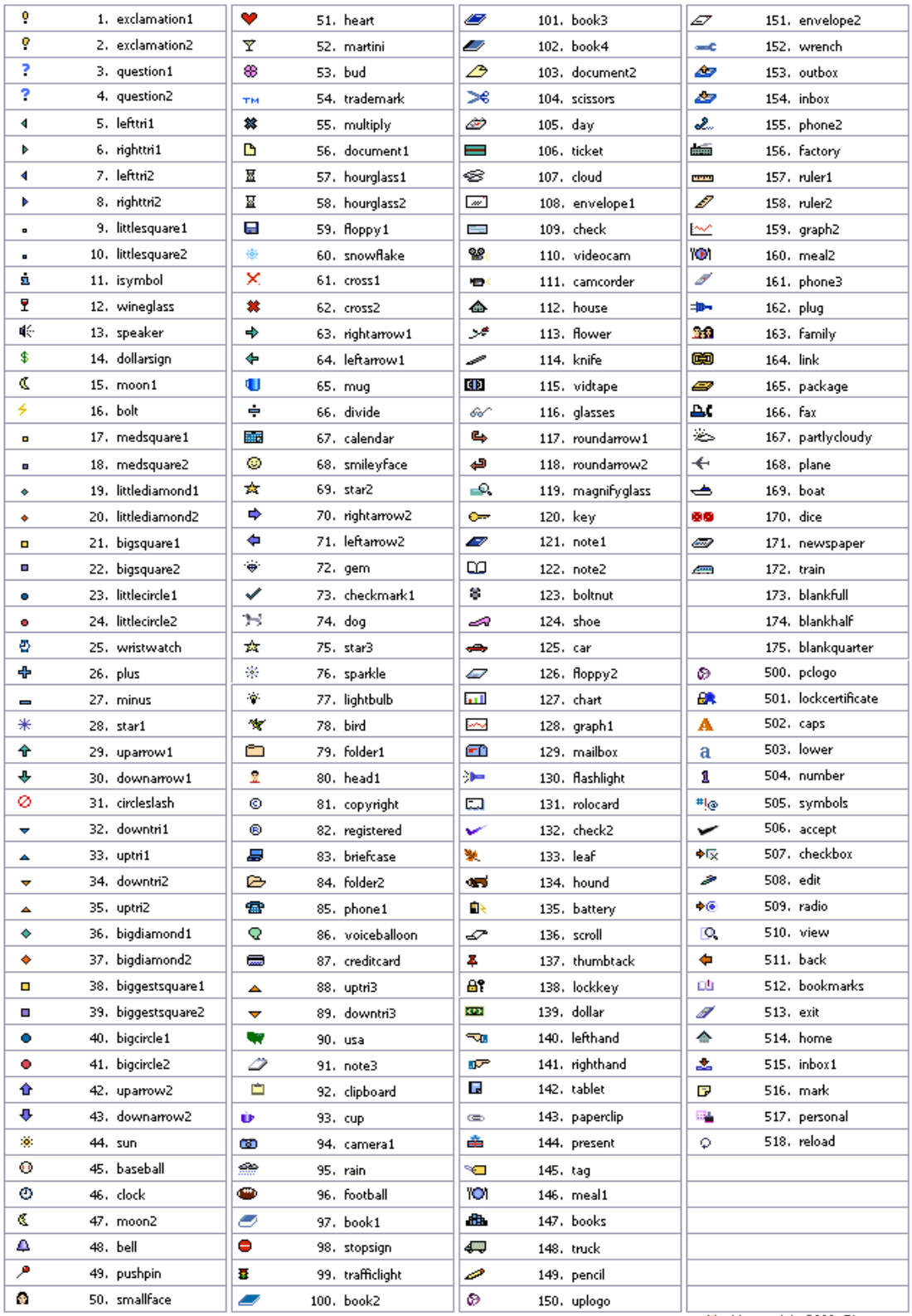
Syntax

```
<img alt="text" src="url" localsrc="icon" align="alignment"
height="n" width="n" vspace="n" hspace="n"/>
```

Attributes

alt	Required. Specifies the text to display if the device does not support images or cannot find the specified image.
src	Required. The URL of the image to display. If you specify a valid icon for the localsrc attribute (see below), the device ignores this attribute.
localsrc	The name of a known icon. If the device cannot find the icon in ROM (Read-Only Memory), it attempts to retrieve it from the Openwave Mobile Access Gateway Server. If you specify a valid icon (see Figure 2-6 for a list of icon names), the device ignores the src and alt attributes (see above) even though they are still required.
align	top | middle | **bottom** Specifies image alignment relative to the current line of text.
height	Specifies the height of the image. If specified as a percentage value, the size is based on the available vertical space, and not on the natural size of the image.
width	Specifies the width of the image. If specified as a percentage value, the size is based on the available horizontal space, and not on the natural size of the image.
vspace	Specifies the amount of white space to be inserted above and below the image. If specified as a percentage value, the space is based on the available vertical space, and not on the natural size of the image.
hspace	Specifies the amount of space to the left and right of the image. The default setting is zero.

Figure 2-6. Icon names

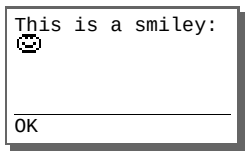


Listing 2-8. I<img> element

The following WML deck generates the display shown in Figure 2-7:

```
<wml>
<card>
  <p>
    This is a smileyface:
    <br/><img alt=":-)" localsrc="smileyface" src=""/>
  </p>
</card>
</wml>
```

Figure 2-7. Card containing localsrc icon



NOTE Some localsrc icons have different forms when displayed at different font sizes. Since you cannot control the display size, keep in mind that the appearance of these icons may vary slightly on different devices.

<input>

The <input> element lets the user enter text which the device assigns to a specified variable.

Syntax

```
text
<input name="variable" title="label" type="type"
value="value" format="specifier"
emptyok="boolean" size="n" maxlength="n" tabindex="n"/>
```

where *text* represents the text and/or image the device will display to prompt the user for entry.

Attributes

name	Required. The name of the variable in which the device stores the text entered by the user. When the device displays the <input> element, the value in the specified variable appears in the entry field.
title	Specifies a brief label for the input item. Some devices use the label as a tooltip when displaying the input field. Others might use it as a label for a user interface mechanism that lets the user navigate to the item. For example, if a device cannot display all card content on one screen and ordered="true" (see "Order options" on page 17), the Mobile Browser uses the title to identify this input item on a summary-level menu.
type	**text** | password Specifies how the device should display text the user enters. Specifying type="text" causes the text to be visible. Specifying type="password" causes the text to be masked (for example, replaced by "*" characters). Note that the password mode is not *encrypted* so you should not rely on it for securing critical data.
value	Specifies the value of the variable named in the name attribute. When the element is displayed and the variable named in the name attribute is not set, the name variable is assigned the value specified in the value attribute. If the name variable already contains a value, the value attribute is ignored. If the value attribute specifies a value that does not conform to the input mask specified by the format attribute, the user agent must ignore the value attribute.
accesskey	A number (0-9) that appears on the left side of the screen next to the link. If the user presses the corresponding key on the phone keypad, the phone executes the task defined by the link. We recommend that you number the links in the order in which they appear.

`format`	Specifies a data format that the user entry must match (see "Specifying a Format Mask" below). If you omit this attribute, the device assumes `*M` (default uppercase first character followed by up to `maxlength` number of mixed case alphabetic and numeric characters).
`emptyok`	**`true`** | `false` Specifies whether the user can leave the field blank. Specifying `emptyok="true"` indicates that the field is optional—if the user enters a value, however, the device applies any entry requirements you specify for the `format` attribute (see above).
`size`	Controls how large on screen the input field will be rendered. If the `size` attribute is not included, the field will grow to accommodate all of the characters that the user enters. If a value is given for the `size` attribute, when the user enters more characters than can be displayed in the field, the characters scroll off the screen to the left. When the user navigates off of the input element, only the beginning of the input will be visible in the field.
`maxlength`	Specifies the maximum number of characters the user can enter. If you do not specify the `maxlength` attribute, the Mobile Browser imposes a limit of 256 characters.

Specifying a Format Mask

You can specify the following values for the `format` attribute:

Tag	Description
`A`	Any symbolic or uppercase alphabetic character (no numbers)
`a`	Any symbolic or lowercase alphabetic character (no numbers)
`N`	Any numeric character (no symbols or alphabetic characters)
`X`	Any symbolic, numeric, or uppercase alphabetic character (*not changeable* to lowercase)
`x`	Any symbolic, numeric, or lowercase alphabetic character (*not changeable* to uppercase)
`M`	Any symbolic, numeric, or uppercase alphabetic character (*changeable* to lowercase)—for multiple character input, defaults to uppercase first character
`m`	Any symbolic, numeric, or lowercase alphabetic character (*changeable* to uppercase)—for multiple character input, defaults to lowercase first character

- To limit the number of characters users can enter, you can specify a single digit number before the character tag—for example, `format="3X"` lets the user enter a maximum of three symbolic, numeric, or uppercase alphabetic characters.
- To let users enter an unlimited number of characters, specify an asterisk (`*`) before the character tag—for example, `format="*a"` lets the user enter any number of symbolic or lowercase alphabetic characters.

Listing 2-9. `<input>` **element**

```
<wml>
  <card>
   <p>
      First Name:
      <input name="fname" maxlength="15" /> <br/>
      Last Name:
      <input name="lname" maxlength="15" /> <br/>
      State:
      <input name="state" maxlength="2" emptyok="true"
         value="CA" /> <br/>
      Zipcode:
      <input name="zipcode" maxlength="10" /> <br/>
      Password:
      <input name="password" maxlength="8" type="password"
        />
   </p>
  </card>
</wml>
```

<link>

The <link> element specifies a relationship between the containing deck and another document. This element must exist inside the <head> element.

Syntax

```
<wml>
  <head>
    <link href="/next" rel="next"/>
  </head>
. . .
</wml>
```

Attributes

href	Required. Specifies the location of the document being linked to.
rel	Required. Specifies the relationship between this deck and the document referenced by the href attribute.
sendreferer	true | **false** Specifies whether the device should include the deck URL in the URL request. Specifying sendreferer="true" causes the device to set the HTTP_REFERER header to the *relative* URL of the requesting deck. If you want to restrict access to trusted services, decks that request specified URLs must set this option to TRUE.

Listing 2-10. <link> **element**

The following example demonstrates how to preload WML pages into the Browser cache using the <link> element. This can provide a faster user experience while navigating. If the user clears and views the cache before loading the contents.wml file, they will see no URLs present in the cache. After contents.wml is loaded, then if the user views the cache again, they will see contents.wml, chap1.wml, chap2.wml, and appendix.wml all in cache. Subsequent navigation to chap1, chap2, and appendix all result in a faster view because of cache hits.

```
CONTENTS.WML
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//OPENWAVE.COM//DTD WML 1.3//EN"
"http://www.openwave.com/dtd/wml13.dtd" >
<wml>
    <head>
      <link href="chap1.wml" rel="next"/>
      <link href="chap2.wml" rel="next"/>
      <link href="appendix.wml" rel="next"/>
     </head>
   <card id="main">
      <p>
         Table of Contents:
         <a href="chap1.wml" title="CHAP1">Chapter1</a><br/>
         <a href="chap2.wml" title="CHAP2">Chapter2</a><br/>
         <a href="appendix.wml"
          title="APPND">Appendix</a><br/>
      </p>
   </card>
</wml>
CHAP1.WML
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//OPENWAVE.COM//DTD WML 1.3//EN"
"http://www.openwave.com/dtd/wml13.dtd" >
<wml>
   <head>
      <link href="contents.wml" rel="next"/>
      <link href="chap2.wml" rel="next"/>
   </head>
   <card id="main">
      <do type="accept" label="CHAP2">
         <go href="chap2.wml"/>
      </do>
      <do type="options" label="TOP">
         <go href="contents.wml"/>
      </do>
      <p>Chapter 1</p>
   </card>
</wml>
```

```
CHAP2.WML
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//OPENWAVE.COM//DTD WML 1.3//EN"
"http://www.openwave.com/dtd/wml13.dtd" >
<wml>
    <head>
      <link href="contents.wml" rel="next"/>
      <link href="appendix.wml" rel="next"/>
    </head>
    <card id="main">
      <do type="accept" label="APPDX">
         <go href="appendix.wml"/>
      </do>
      <do type="options" label="TOP">
         <go href="contents.wml"/>
      </do>
         <p>Chapter 2</p>
    </card>
</wml>
APPENDIX.WML
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//OPENWAVE.COM//DTD WML 1.3//EN"
"http://www.openwave.com/dtd/wml13.dtd" >
<wml>
    <head>
      <link href="contents.wml" rel="next"/>
    </head>
   <card id="main">
   <do type="accept" label="TOP">
      <go href="contents.wml"/>
   </do>
      <p>
         Appendix
      </p>
   </card>
</wml>
```

<meta>

The <meta> element provides meta information for a WML deck. This element is specified within the deck header along with any access control information for the deck (for more information, see "<access>" on page 11 and "<head>" on page 29). Note that not all devices support every meta information type.

Syntax

```
<head>
    <meta http-equiv="Cache-Control" content="max-age=time"
    forua= true />
    ...
</head>
```

Attributes

property	Required. You must specify *one* of the following attributes: name="*name*" http-equiv="*name*" • If you specify the name attribute, the Openwave Mobile Access Gateway Server ignores the metadata. • If you specify the http-equiv attribute, the Openwave Mobile Access Gateway Server converts the metadata to an HTTP response header.
content	Required. Specifies the metadata value associated with the *property* attribute.
scheme	The Mobile Browser software does *not* currently support this attribute.
forua	**true** | false Required. Specifies that the author intended the property to reach the user agent. If forua="false", an intermediate agent must remove the <meta> element before the document is sent to the client. If the value is "true", the metadata of the element must be delivered to the user agent. The method of delivery may vary. For example, http-equiv metadata may be delivered using HTTP or WSP headers.

Cache Control

Like conventional Web browsers, the mobile browser device has a memory cache. It caches each deck that the user visits in order to quickly redisplay it without requesting it from the Openwave Mobile Access Gateway Server again. The length of time that a device keeps a deck in cache is called the *time to live* (TTL). The default mobile browser device TTL is 30 days (or until memory is exhausted). If a deck contains time-sensitive information, you can specify a shorter TTL so that the device will reload the deck from the server more frequently. The following example illustrates how to use a <meta> statement to set the TTL:

```
<meta http-equiv="Cache-Control" content="max-age=3600"
   forua= true/>
```

The max-age parameter specifies the time (in seconds) to cache the deck. The example above instructs the device to drop the deck from the cache after 1 hour (3600 seconds). To determine the right TTL for a deck, you should balance the time-sensitivity of the information with the degradation in response time caused by reloading information from the server. Setting max-age to zero causes the Mobile Browser to reload the deck every time the user navigates to it in a forward direction; however, if the user navigates *back* to the deck, the Mobile Browser displays the card from the information in cache.

The Browser now performs "if-modified-since" content negotiation with the HTTP server, but only after a deck's TTL has expired. Decks may now specify "no-cache" in the Cache-control <meta> element, which is equivalent to "max-age=0." Decks may also specify "must-revalidate" in the Cache-control <meta> element, which forces the Browser to revalidate the deck's TTL, even if the user navigates to the deck in the backward direction. Finally, if the Cache-control <meta> tag is not defined in the WML deck, the browser may derive the deck's TTL from the HTTP cache headers, based on the HTTP/1.1 caching model.

```
<meta http-equiv="Cache-Control" content="must-revalidate"
   forua= true/>
<meta http-equiv="Cache-Control" content="no-cache"
   forua= true/>
```

Bookmarks

Mobile browser device bookmarks are similar to conventional Web browser bookmarks. When a user bookmarks a card, the Mobile Browser creates a bookmark that consists of two items:

- A label that identifies the bookmark—this label comes from the title attribute within the <card> definition (see "<card>" on page 16).
- The URL to open when the user selects the bookmark.

Because all decks are now bookmarkable by default, the markable <meta> element is only used to turn off bookmarking for a deck. The syntax for the statement is as follows:

```
<meta name="vnd.up.markable" forua="true" content="false"/>
```

When a user bookmarks a card, the Mobile Browser automatically sets the bookmark URL to the URL for the deck. If you want to use a different URL, you can specify another <meta> element in the deck header using the following syntax:

```
<meta name="vnd.up.bookmark" forua="true" content="url"/>
```

where *url* is the URL you want to use for the bookmark.

<noop>

The <noop> element is a *task* element that instructs the device to do nothing, for example, "no operation." This element is useful for overriding deck-level <do> elements, called *shadowing* (see the *WML 1.3 Developer's Guide* for more information on shadowing).

Syntax

```
<noop/>
```

<onevent>

The <onevent> element associates a state transition, or *intrinsic event*, with a task. When the intrinsic event occurs, the device performs the associated <onevent> task.

Syntax

```
<onevent type="type">task</onevent>
```

where *task* represents the action to perform when the intrinsic event occurs:

task	You can specify any *one* of the following actions for the <onevent> element: • <go> (see "<go>" on page 27) • <prev> (see "<prev>" on page 50) • <noop> (see "<noop>" on page 43) • <refresh> (see "<refresh>" on page 53) • <exit> (see "<exit>" on page 25) • <spawn> (see "<spawn>" on page 61) • <throw> (see "<throw>" on page 68)

Attributes

type	Required. Identifies the intrinsic event that triggers the specified <onevent> task (see descriptions below). If a *card-level* <onevent> element (for example, defined within a <card> element) has the same type as a *deck-level* <onevent> element (for example, defined within a <template> element), the card-level binding overrides the deck-level binding.

You can specify the following values for the type attribute:

type value	Perform task if ...
onpick	User selects or deselects an <option> item (see "<option>" on page 46).
onenterforward	User navigates to a card through a <go> task.
onenterbackward	User navigates to a card through a <prev> task or invokes the PREV mechanism (for example, presses the BACK key).
ontimer	A specified <timer> element expires (see "<timer>" on page 69).

<optgroup>

The <optgroup> element allows you to group multiple <option> (or nested <optgroup>) elements within a card. Creating option groups lets you specify control information about how the device should present the card content.

Syntax

```
<optgroup label="label">content</optgroup>
```

where *content* represents one or more of the following:

content	You can specify any of the following elements: • <optgroup> (a nested <optgroup> element) • <option> (see "<option>")

Devices display these elements in the order in which you specify them.

Attributes

label	Specifies a brief label for the <optgroup> group. Some devices use the label as a title when displaying the <optgroup> content. Others might use it as a label for a user interface mechanism that lets the user navigate to the <optgroup> content.

<option>

The <option> element specifies a particular choice within a <select> element.

IMPORTANT The <option> element has been extended to allow an image in the <option> text flow. When including an image in the <option> element, the image attributes src and alt are both required. (See "<img>" on page 32.) For example:

```
<option value="1">
    <img src="/img1" localsrc="" alt="Choice 1"/>
</option>
```

Syntax

```
<option title="label" value="value" onpick="url">
    content
</option>
```

where *content* represents the text the device will display to represent the particular selection item and the action to perform if the user selects it:

event	You can optionally specify the following element in your item definition: • <onevent> (see "<onevent>" on page 44)
text	The device displays this text to represent the selection item.

Attributes

title	A label that identifies the option. The Mobile Browser uses the title as the ACCEPT key label when the user selects the option. To ensure compatibility on a wide range of devices, *label* should be a maximum of five characters.
value	Specifies the value to assign to the variable defined in the <select> element name attribute if the user selects the option (see example). If you specify a variable reference, the device evaluates the reference before setting the name variable.
onpick	Specifies the URL to open if the user selects the option (*or* deselects it if the <select> element allows multiple choices). This attribute represents an abbreviated form of the <onevent> element.

Listing 2-11. `<option>` **element**

In the following example, selecting an item sets the variable `color` to the value associated with that item (for example, `1`, `2`, `3`, or `4`).

```
<wml>
<card>
  <p>
    Please select your favorite color:
    <select name="color">
        <option value="1">red</option>
        <option value="2">blue</option>
        <option value="3">green</option>
        <option value="4">yellow</option>
    </select>
  </p>
</card>
</wml>
```

<p>

The <p> element specifies a new paragraph and has alignment and line-wrapping attributes.

Syntax

```
<p align="alignment" mode="wrapmode">
   content
</p>
```

Attributes

align	**left** | right | center Specifies line alignment relative to the display area. Specifying <p> without the align attribute resets the line to left alignment.
mode	**wrap** | nowrap Specifies text wrapping mode to use. If you specify nowrap, the device uses another mechanism, such as horizontal scrolling, to display long lines to the user. The device continues to use the mode you specify until you specify a <p> element with the other mode. In other words, • If you specify the mode attribute, the device applies that mode. • If you do not specify the mode attribute, the device applies the last-specified mode value. If no previous <p> element exists, the device applies the default mode (wrap).

<postfield>

The <postfield> element defines name/value pairs that are passed to the HTTP server receiving the <go> request. See the "<go>" on page 27 for an example of the <postfield> element's use in WML.

Syntax

```
<postfield name="name" value="value"/>
```

Attributes

name	Required. A label that identifies the field.
value	Required. A string specifying the default value for the variable specified by the value attribute.

<pre>

The <pre> element tells visual user agents that the enclosed text is preformatted. When handling preformatted text, user agents may:

- Leave white space intact.
- Render text with a fixed-pitch font.
- Disable automatic word wrap.

Syntax

```
<pre>
  content
</pre>
```

content can be any combination of text or the elements:

```
a, anchor, do, u, br, i, b, em, strong, input, select
```

Attributes

```
optional id
class
```

As with any other WML element.

<prev>

The <prev> element is a *task* element that instructs the device to remove the current URL from the history stack and open the previous URL. If no previous URL exists on the history stack, specifying <prev> has no effect.

Syntax

```
<prev>content</prev>
```

where *content* represents the variables to set when opening the previous URL:

content	You can optionally specify one or more variables in a <prev> statement: • <setvar> (see "<setvar>" on page 60)

Listing 2-12. <prev> **element**

```
<wml>
<head>
  <meta  forua="true" content="true"/>
</head>
<!--This is the first card-->
<card  id="card1">
  <onevent  type="onenterforward">
    <go  href="#card2">
      <setvar  name="Var1" value="Telephone"/>
    </go>
  </onevent>
  <do  type="accept" label="MENU">
    <go  href="prevIndex.wml"/>
  </do>
<p>
Card 1 <br/>
Var1 = $(Var1). <br/>
Click Menu to return to the test index.
</p>
</card>
<!--This is the second card-->
<card  id="card2">
<p>
  <do  type="accept" label="PREV">
   <prev>
     <setvar  name="Var1" value="32"/>
   </prev>
  </do>
Card 2 <br/>
Var1 = $(Var1). <br/>
Click PREV to go to Card 1.
</p>
</card>
</wml>
```

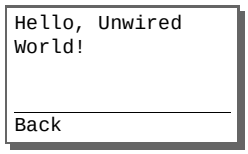
IMPORTANT Unlike other WML elements that have content, specifying content for the <prev> element is optional. If you do *not* specify any content, you must use the syntax <prev/> rather than <prev>*content*</prev>.

Listing 2-13. <prev> **element without content**

The following WML illustrates a <prev> element that does not contain any content.

```
<wml>
<card>
    <do type="accept" label="Back">
        <prev/>
    </do>
  <p>
Hello, Unwired World!
  </p>
</card>
</wml>
```

Figure 2-8. Card with ACCEPT **key bound to** <prev> **task**



<receive>

The <receive> element is used to receive data sent from a child context. A <receive> element without a name attribute causes the value in the parameter block to be ignored.

When receiving a parameter block, <receive> elements assign a corresponding variable to each value in that parameter block. If there are insufficient values in the parameter block, each additional <receive> should be treated as if the parameter block contained an empty string in that position.

Syntax

```
<receive name="X"/>
```

Attributes

name	Specifies the variable name. It is an error if the name attribute value is not a legal WML variable name.

NOTE See “<send>” on page 58 for example code.

<refresh>

The <refresh> element is a *task* element that instructs the device to refresh the specified card variables. The device also refreshes the display if any of those variables are currently shown.

Syntax

```
<refresh>content</refresh>
```

where *content* represents the variables to refresh:

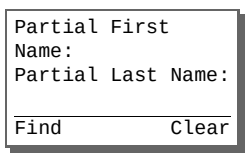
content	You must specify at least one variable in a <refresh> statement: • <var> (see "<setvar>" on page 60)

Listing 2-14. <refresh> element

The following WML builds on the example shown on page 28 by adding a "Clear" option that lets the user clear the First Name and Last Name fields in order to enter new search criteria. The code specifies a second <do> element that uses <refresh> to reset the first and last variables to NULL when the user presses the OPTIONS key. Figure 2-9 illustrates how the card appears on the device.

```
<wml>
<card title="Employee" ordered="true">
    <do type="ACCEPT" label="Find">
        <go method="post" href="?next=list">
        <postfield name="NEXT" value="list"/>
        <postfield name="L" value="$last"/>
        <postfield name="F" value="$first"/>
        </go>
    </do>
    <do type="OPTIONS" label="Clear">
        <refresh>
            <setvar name="first" value=""/>
            <setvar name="last" value=""/>
        </refresh>
    </do>
    <p>
    Partial First Name:
    <input title="First Name" name="first"/>
    Partial Last Name:
    <input title="Last Name" name="last"/>
    </p>
</card>
</wml>
```

Figure 2-9. Card with OPTIONS key bound to <refresh> task



<reset>

The <reset> element causes all variables in the current context to be cleared. If a task element, <go>, <prev>, or <refresh> contains a <reset> element, the reset operation is performed when the task is executed. If the <catch> element contains a <reset> element, the operation is performed during the <throw> task processing.

Syntax

```
<go href="/bar">
   <reset/>
</go>
```

For example, if a <go> element includes a <reset>, all variables in the context would be unset as a result of executing the <go> task.

Listing 2-15. <reset> **element**

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//OPENWAVE.COM//DTD WML 1.3//EN"
"http://www.openwave.com/dtd/wml13.dtd" >
<wml>
   <card id="main">
      <do type="accept" label="Get">
         <refresh>
            <setvar name="weather" value="Sunny"/>
            <setvar name="temp" value="72"/>
            <setvar name="wind" value="8 mph"/>
            <setvar name="humidity" value="32%"/>
         </refresh>
      </do>
      <do type="options" label="Clear">
         <refresh>
            <reset/>
         </refresh>
      </do>
      <p>
         Weather: $(weather) <br/>
         Temp: $(temp) <br/>
         Wind: $(wind) <br/>
         Humidity: $(humidity)
      </p>
   </card>
</wml>
```

<select>

The <select> element specifies a list of options from which the user can choose. You can specify either single- or multiple-choice <select> elements.

Syntax

```
text
<select title="label" multiple="boolean" name="variable"
value="default" iname="index_var" ivalue="default"
tabindex="n">
    content
</select>
```

where *text* represents the text and/or image the device will display to prompt the user for selection and *content* represents the list of items from which to choose. You can define selection *content* using one or more of the following elements:

content	You can specify either of the following elements in a <select> list: • <optgroup> • <option>

Devices display these elements in the order in which you specify them.

Attributes

title	Specifies a brief label for the <select> list. Some devices use the label as a title when displaying the <select> content. Others might use it as a label for a user interface mechanism that lets the user navigate to the <select> content. For example, if a device cannot display all card content on one screen and ordered="true" (see "Order options" on page 17), the Mobile Browser uses the title to identify this select list on a summary-level menu.
multiple	true | **false** Specifies whether the user can select multiple items.
name	The name of the variable in which the device stores the value(s) associated with the option(s) chosen by the user. The value associated with each option comes from the <option> element value attribute. The value(s) in the specified variable determine the default selection(s) when the device displays the <select> element. If the variable has no value, the device sets it to the value(s) specified for the default attribute. If you do not specify a default value, the device initializes the variable to an empty string (""). In the case of multiple selections, the values are stored as a semicolon-separated list (see example).

value	A string specifying the default value(s) for the variable specified by the name attribute. If the name attribute already has a value when the user navigates to the <select> element, the device ignores the value attribute. If the name attribute does not already have a value, the device sets it to the value specified by the value attribute.
iname	Identical to the name attribute except for the following: • The specified variable stores the *index* value(s) associated with the option(s) chosen by the user. The index value associated with each option comes from its *position* in the <select> list, starting with 1. If the user has not selected an option, the index value is 0. • The default value is specified by the ivalue attribute.
ivalue	Identical to the default attribute except for the following: • The specified string contains the default *index* value(s) for the variable specified by the iname attribute.
list	Each <option> in the select element will render on screen with a key accelerator (1 through 9) to the left of the text that is enclosed by the <option></option> tags.
popup	Renders a single <option> on the screen of the device until the <select> element is activated. When the select element is activated, a popup appears on the screen. The up/down navigation tool is used to highlight the desired option. When the user presses the <accept> key, the popup is removed from the display and the selected item now is rendered on the screen.
radio	Renders all of the <options> on the screen at the same time. Each option is preceded on the screen by a radio button. Only one <option> can have its radio button selected at any give time.
spin	Renders only a single <options> on the screen of the device at a given time. When the user activates the select element by highlighting it and pressing the accept key, the up/down navigation serves to scroll through each of the <options> within the select. The user can navigate up or down through the list, and the list is presented as a continuous loop (for example, the item after the last item in the list is the first item).
tabindex	The Mobile Browser software does *not* currently support this attribute.

NOTE If the type attribute is not specified, type=list is implied.

Listing 2-16. `<select>` **element**

In the following example, “Dog” and “Cat” are the default selections if the variable `i` has no value when the device displays the card. Choosing “Cat” and “Horse” sets the variable `x` to the value `"C;H"` and the variable `i` to the value `"2;3"`.

```
<wml>
<card>
  <p>
    Please choose your favorite animals:
    <select multiple="true" name="x" iname="i" ivalue="1;2">
        <option value="D">Dog</option>
        <option value="C">Cat</option>
        <option value="H">Horse</option>
    </select>
  </p>
    <do type="accept"><noop/></do>
</card>
</wml>
```

<send>

The <send> element specifies a single value to be included in a parameter block.

The Browser creates a parameter block with a single entry for each <send> element. Each entry is identified by its position in the parameter block, and the position is derived from the order of the <send> elements.

Syntax

```
<send value="$X99"/>
<send value=”$X100”/>
```

Attributes

value	Specifies the data to be sent in this parameter block position. If not specified, the value defaults to an empty string.

NOTE See “<spawn>” on page 61 for additional example code.

Listing 2-17. <send> **element**

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//OPENWAVE.COM//DTD WML 1.3//EN"
"http://www.openwave.com/dtd/wml13.dtd" >
<wml>
 <card id="main">
  <do type="accept" label="Get">
   <spawn href="#GetErrorCode">
     <receive name="errorcode"/>
     <receive name="errorname"/>
   </spawn>
  </do>
  <do type="options" label="Clear">
   <spawn href="#ClearErrorCode">
     <receive name="errorcode"/>
     <receive name="errorname"/>
   </spawn>
  </do>
  <p>
    Error Codes: <br/>
    ErrorCode: $(errorcode) <br/>
    ErrorName: $(errorname)
  </p>
 </card>
 <card id="GetErrorCode">
  <do type="accept"
   <exit>
     <send value="99"/>
     <send value="Inv. Key"/>
   </exit>
  </do>
  <p> Return Error Codes</p>
 </card>
 <card id="ClearErrorCode">
  <do type="accept">
   <exit>
     <send value=""/>
     <send value=""/>
   </exit>
  </do>
  <p> Clear Error Codes</p>
 </card>
</wml>
```

<setvar>

The <setvar> element sets a variable to a specified value when the device executes a <go>, <prev>, <spawn>, or <refresh> task.

Syntax

```
<setvar name="name" value="value"/>
```

Attributes

name	Required. The name of the variable to set. The device ignores the <setvar> element if *name* does not evaluate to a known variable at runtime.
value	Required. The value to assign to the variable.

<small>

The <small> element specifies small font text.

Syntax

```
<small>text</small>
```

where *text* is the text to display in small font.

IMPORTANT Support for this element is available when using Release 4.0 or later of the Openwave Browser.

<spawn>

The <spawn> element declares a spawn task, indicating the creation of a child context and invocation of a URL in that child context. If the URL names a WML card or deck, the card is displayed, and the URL becomes the basis for a new history stack in the child context.

When the child context is exited via the exit task, an onexit intrinsic event occurs. The onexit event can be handled with the onexit attribute or by embedding an <onevent> inside the <spawn> element. A spawn task can initialize the child context's variables with the <setvar> element, parameters returned from the child context are bound to variables with <receive> elements, and exceptions that occur in child contexts can be caught with the <catch> element.

The spawn element may also contain one or more <postfield> elements. These elements specify information to be submitted to the origin server during the request.

Syntax

```
<spawn href="/child" onexit="/continue">
   <setvar name="Name" value="Joe"/>
</spawn>
```

The <spawn> element creates a new child context, and invokes the “/child” URL in this new context. The child context initializes with the variable “*Name*” which evaluates to “Joe”. When the child context exits, the onexit event occurs, resulting in a <go> task to the “/continue” URL.

Attributes

`href`	Required. Specifies the destination URL. The URL of the card to display.
`onexit`	The `onexit` event occurs when the child context is exited with an exit task.
`sendreferer`	`true` | **`false`** Specifies whether the device should include the deck URL in the URL request. Specifying `sendreferer="true"` causes the device to set the `HTTP_REFERER` header to the *relative* URL of the requesting deck. If you want to restrict access to trusted services, decks that request specified URLs must set this option to `TRUE`.
`method`	**`get`** | `post` Specifies the HTTP submission method. Specifying `method="post"` causes the Openwave Mobile Access Gateway Server to transcode variable data to the character set specified by the HTTP headers defined in your application. You should perform this transcoding if non-ASCII characters (specifically `UTF-8`) may exist in the data being passed. For more information on character sets and HTTP headers, see the *WML 1.3 Developer's Guide*. If you do *not* specify the `method` attribute, the device automatically uses the `get` method.
`accept-charset`	Specifies the character encodings that your application can handle. The device uses this attribute to transcode data specified by the `postfield element`. The Openwave Mobile Access Gateway Server assumes `UTF-8` as the default encoding (of which `US-ASCII` is a subset), so WML services in the United States, Canada, or Australia do not need to use this attribute. You can also omit this attribute if you specify your character set(s) in the HTTP response header. Note that the `accept-charset` attribute overrides any character encodings you specify in the HTTP header. The syntax for this attribute is a comma- or space-delimited list of IANA character sets. For example, `accept-charset="UTF-8, US-ASCII, ISO-8859-1"`. • For a list of supported encoding names, see the *WML 1.3 Developer's Guide*. To view the complete IANA Character Set registry, go to `http://www.iana.org/`

Listing 2-18. `<spawn>` **element**

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//OPENWAVE.COM//DTD WML 1.3//EN"
        "http://www.openwave.com/dtd/wml13.dtd" >
<!-- Browse Headlines -->
<wml>
  <card title="Headlines">
    <do type="accept">
      <spawn href="$(cardname:noesc)">
        <!-- Catch all exceptions -->
<catch/>
      </spawn>
```

```
    </do>
    <p mode="nowrap">
      Financial News
      <select name="cardname">
        <option value="#n1">Stock Market Up</option>
        <option value="#n2">Fed Raises Interest Rates</option>
        <option value="#n3">Labor Strike Looming</option>
      </select>
    </p>
  </card>
  <card id="n1" title="Story">
    <do type="accept" label="Skip">
      <go href="#n2"/>
    </do>
    <do type="options" label="Done">
      <exit/>
    </do>
    <p>
Stock Market UP
      <br/>----
      <br/>blah blah blah blah blah blah blah blah
    </p>
  </card>
  <card id="n2" title="Story">
    <do type="accept" label="Skip">
      <go href="#n3"/>
    </do>
    <do type="options" label="Done">
      <exit/>
    </do>
    <p>
Fed Raises Interest Rates
      <br/>---
      <br/>blah blah blah blah blah blah blah blah
    </p>
  </card>
  <card id="n3" title="Story">
    <do type="accept" label="Skip">
      <go href="#n1"/>
    </do>
    <do type="options" label="Done">
      <exit/>
    </do>
    <p>
Labor Strike Looming
      <br/>---
      <br/>blah blah blah blah blah blah blah blah
    </p>
  </card>
</wml>
```


The element specifies strongly emphasized text.

Syntax

```
<strong>text</strong>
```

where *text* is the text to display in strongly emphasized font.

IMPORTANT Support for this element is available when using Release 4.0 or later of the Openwave Browser.

<table>

The <table> element allows you to specify columnar format. WML tables are similar to HTML tables. It is used together with the <tr> and <td> elements to create sets of aligned columns of text and images in a card. Nesting of <table> elements is not allowed. The <table> elements determine the structure of the columns.

When defining a table, you have to declare the number of columns, followed by some content. The content can include empty rows and columns. Whilst the elements separate content into columns, they do not specify column or intercolumn widths.

Syntax

```
<table title="name" align="left|right|center" columns="number
of columns">
...row and data declarations
</table>
```

Attributes

align	**left** | right | center Specifies text alignment relative to the column. If you do not specify the align attribute, the text is automatically left-aligned.
title	Specifies a label for the table.
columns	Required. Specifies the number of columns for the row set. Specifying a zero value for this attribute is not allowed.

Listing 2-19. <table> **element**

```
<wml>
    <card>
        <p>
            Prague <img  localsrc="sun" alt="" src=""/>
        </p>
        <p  mode="nowrap">
            <table columns="2">
                <tr><td><i>High</i></td>
                <td><i>Low</i></td></tr>
                <tr><td>53</td>
                <td>42</td></tr>
            </table>
            Today: Mostly sunny and cool.
            <br/>Tonight: Cold with chance of light rain.
        </p>
    </card>
</wml>
```

<td>

The <td> element is used as a container to hold a single table cell data within a table row. Table data cells may be empty. The user agent should do a best effort to deal with multiple line data cells that may result from using images or line breaks.

Syntax

```
<td>content</td>
```

where *content* represents text inside the table cell, or the <img> or <anchor> elements.

content	You can specify either of the following elements in a <td> element: • <img> (see "<img>" on page 32) • <anchor> (see "<anchor>" on page 12)

<tr>

The <tr> element is used as a container to hold a single table row. Table rows may be empty, in other words, all cells are empty.

Syntax

```
<tr>
  <td>content</td>
</tr>
```

where *content* represents text inside the table cell, or the <img> or <anchor> elements.

<template>

A WML deck may contain a <template> element that defines deck-level event bindings, for example, characteristics that apply to all cards in the deck. You can override these characteristics for a particular card by specifying the same event bindings within the <card> definition (see "<card>" on page 16).

Syntax

```
<wml>
<template onenterforward="url" onenterbackward="url"
ontimer="url">
    content
</template>
</wml>
```

where *content* represents the general action to take when particular events occur:

content	You can specify either of the following elements in a template definition: • <do> (see "<do>" on page 22) • <onevent> (see "<onevent>" on page 44)

Attributes

onenter forward	Specifies the URL to open if the user navigates to a card through a <go> task. This attribute represents an abbreviated form of the <onevent> element.
onenter backward	Specifies the URL to open if the user navigates to a card through a <prev> task. This attribute represents an abbreviated form of the <onevent> element.
ontimer	Specifies the URL to open if a specified <timer> element expires. This attribute represents an abbreviated form of the <onevent> element.

<throw>

The <throw> element declares a throw task, indicating that an exception should be raised. Values may be sent to the exception handler with <send> elements included in the throw. Throwing an exception terminates the current context and causes the context to be destroyed, including any variable and history state contained in the context.

If the parent context does not contain an exception handler (a <catch> element) that matches this exception, or a </catch> element, the parent context is terminated and the exception is re-thrown to that context's parent. This operation repeats until an exception handler is found or all parent contexts have been terminated. In the case where all contexts are terminated, the Browser performs a reset to a predictable state. Typically, this clears the history stack and displays the home deck.

For example, the following throws an exception with the name "user input error". In addition, a parameter block is included specifying more information about the error.

Syntax

```
<throw name="user input error">
   <send value="Bad numeric value"/>
</throw>
```

Attributes

name	Required. Specifies the name of the exception. This name is used to find the correct handler for the exception. The name attribute value is case sensitive.

NOTE See "<catch>" on page 20 for example code.

<timer>

The <timer> element provides a method for invoking a task automatically after some period of user inactivity. Any task or user action that activates the card starts the timer, and executing any task element stops it. You can only associate one task for each timer, and you can only define one timer for each card.

Syntax

```
<timer name="variable" value="value"/>
```

Attributes

name	The name of the variable in which the device stores the timer value. If the variable has no value when the timer is initialized, the device sets it to the value specified for the default attribute. The device sets this variable to either the current timer value when the user exits the card or 0 if the timer expires.
value	A string specifying the value for the variable specified by the key attribute. You must specify <timer> values in units of 1/10 seconds—so, for example, a value of 100 equals 10 seconds. Specifying a value of 0 disables the timer. If the name attribute already has a value when the timer is initialized, the device ignores the default attribute. If the name attribute does not already have a value, the device sets it to the value specified by the value attribute.

Listing 2-20. <timer> **element**

The following example illustrates how a timer can initialize and reuse a counter. The device resets the timer to the value of the time variable each time the user navigates to the card. If time has no value, the device sets the timer to 5 seconds. When the timer expires, the device automatically displays the second card in the deck.

```
<wml>
<card ontimer="#card2">
    <timer name="time" value="50"/>
  <p>
    Hello, Unwired World!
  </p>
</card>
<card id="card2">
    ...
    ...
</card>
</wml>
```

<u>

The <u> element specifies underlined text.

Syntax

```
<u>text</u>
```

where *text* is the text to display in underlined font.

IMPORTANT Support for this element is available when using Release 4.0 or later of the Openwave Browser.

<wml>

The <wml> element specifies a WML deck.

Syntax

```
<wml xml:lang="lang">
<card>
    content
</card>
</wml>
```

where *content* represents the WML elements that define the actions of the deck.

Attributes

xml:lang	Specifies the natural or formal language for the WML document. Specifying the xml:lang attribute overrides any other language specification for the document. See the formal XML specification "Extensible Markup Language (XML), W3C Proposed Recommendation," at http://www.w3.org for more information about specifying a language value. The Mobile Browser software does *not* currently support this attribute.

Listing 2-21. <wml> **element**

```
<wml xml:lang="en-us">
<card>
    ...
    ...
</card>
</wml>
```

3 WML Quick Reference

Decks and Cards

Element	Syntax
<**wml**>	<wml xml:lang="*lang*" > *content* </wml>

<**card**>	<card id="*name*"
	  title="*label*"
	  newcontext="*boolean*"
	  style="*style*"
	  onenterforward="*url*"
	  onenterbackward="*url*"
	  ontimer="*url*" >
	  content
	</card>

<**template**>	<template onenterforward="*url*"
	  onenterbackward="*url*"
	  ontimer="*url*" >
	  content
	</template>

<**head**>	<head> *content* </head>
<**access**>	<access domain="*domain*" path="*path*" />

<**meta**>	<meta name="*name*"|http-equiv="*name*"
	  content="*value*"
	  forua="true|**false**" />

Timers

Element	Syntax
<**timer**>	<timer name="*variable*" value="*value*" />

Variables

Element	Syntax
<**setvar**>	<setvar name="*name*" value="*value*" />

Anchored Links

Element	Syntax
<**anchor**>	<anchor title="*label*">*task text*</anchor>
<**a**>	 *task* *text*

Events

Element	Syntax
<**do**>	<do type="*type*" label="*label*" name="*name*" optional="*boolean*" > *task* </do>
<**onevent**>	<onevent type="*type*" > *task* </onevent>

Tasks

Element	Syntax
<**go**>	<go href="*url*" sendreferer="*boolean*" method="*method*" accept-charset="*charset*" *content* </go>
<**prev**>	<prev> *content* </prev>
<**noop**>	<noop/>
<**refresh**>	<refresh> *content* </refresh>

Openwave Extensions

Element	Syntax
<**catch**>	<catch name=*"error#1"* onthrow=*"/displayError"*> <receive name=*"Msg"*/> </catch>
<**exit**>	<exit> <send value=*"393"*/> <send value=*"$X"*/> </exit>
<**hr/**>	<hr size="3" width="75%"/>
<**link**>	<wml> <head> <link href=*"/next"* rel=*"next"*/> </head> . . . </wml>
<**receive**>	<receive name=*"X"*/>
<**reset**>	<go href=*"/bar"*> <reset/> </go>
<**send**>	<send value=*"$X99"*/>
<**spawn**>	<spawn href=*"/child"* onexit=*"/continue"*> <setvar name=*"Name"* value=*"Joe"*/> </spawn>
<**throw**>	<throw name=*"user input error"*> <send value=*"Bad numeric value"*/> </throw>

Images

Element	Syntax

<**img**>	<img alt="*text*"
	  src="*url*"
	  localsrc="*icon*"
	  align="*alignment*"
	  height="*n*"
	  width="*n*"
	  vspace="*n*"
	  hspace="*n*" />

User Input

Element	Syntax
<**input**>	<input name="*variable*" title="*label*" type="*type*" value="*value*" default="*default*" format="*specifier*" emptyok="*boolean*" size="*n*" maxlength="*n*" tabindex="*n*" />
<**select**>	<select title="*label*" multiple="*boolean*" name="*variable*" default="*default*" iname="*index_var*" ivalue="*default*" tabindex="*n*" > *content* </select>
<**option**>	<option title="*label*" value="*value*" onpick="*url*" > *content* </option>
<**optgroup**>	<optgroup title="*label*" > *content* </optgroup>
<**fieldset**>	<fieldset title="*label*"> *content* </fieldset>

Layout and Text Formatting

Element	Syntax
<**b**>	 *text*
<**big**>	<big> *text* </big>
<**br**>	
<**em**>	 *text*
<**i**>	 *text*
<**p**>	<p align="*alignment*" mode="*wrapmode*" />
<**small**>	 *text*
<**strong**>	 *text*
<**table**>	<table align="*alignment*" title="*label*" columns="*n*"/>
<**td**>	<td>*content*</td>
<**tr**>	<tr> <td>*content*</td> </tr>
<**u**>	<u> *text* </u>

Special Characters

Element	Display character
`<`	< (less than)
`>`	> (greater than)
`'`	' (apostrophe)
`"`	" (quote)
`&`	& (ampersand)
`$$`	$ (dollar sign)
` `	Nonbreaking space
``	Soft hyphen

Index

Symbols

$$ (dollar sign) 7
& (ampersand) 7
' (apostrophe) 7
> (greater than) 7
< (less than) 7
  (non-breaking space) 7
" (quote) 7
 (soft hyphen) 7, 77

A

accept-charset attribute
  in spawn element 62
accept-charset attribute (go element) 28
access element 11, 29, 41
accesskey attribute
  in a element 10
  in anchor element 12
  in input element 35
a element 16
align attribute
  in img element 32
  in p element 48
  in table element 65
alt attribute (img element) 30, 32
ampersand character (&), specifying in text 7
anchor element 12, 66
anchor example 13
apostrophe character (’), specifying in text 7
attributes, syntax for 6

B

b element 14
b example 14
big element 15
bold example 14
bookmarks, enabling and specifying URL 42
br element 15

C

cache control 41
card element 16, 23, 42, 44, 67
card example 18
case
  in variable names 4
catch element 20
catch example 20
CGI variables
  escaping ampersands 7
character elements 7
character set, for WML 4
columns attribute
  in table element 65
content attribute (meta element) 41

D

DOCTYPE, WML extensions 5
do element 16, 22, 28, 43, 53, 67
  defining multiple do elements 24
dollar sign character ($), specifying in text 7
domain attribute (access element) 11

E

em element 25
emptyok attribute (input element) 36
escaping, special characters 7

Examples
<card> 18
anchor element 13
b 14
bold element 14
card element 18
catch element 20
go element 28
head element 29
i element 31
img element 34
input element 37
link element 38
option element 47
prev element 50
prev element without content 51
refresh element 53
reset element 54
select element 57
send element 59
spawn element 62
table element 65
timer element 69
wml element 70
exit element 12, 22, 25, 44
extensions, Openwave 5
extensions to WML 20, 25, 30, 38, 52, 54, 58, 61, 68

F

fieldset element 16, 17, 26
format attribute (input element) 36
forua attribute (meta element) 41

G

go element 12, 17, 19, 22, 27, 44, 60
go example 28
greater than character (>), specifying in text 7

H

head element 11, 29, 41
head example 29
height attribute (img element) 32
href attribute
in link element 38
in spawn element 62
href attribute (go element) 27
hspace attribute (img element) 32
http-equiv attribute (meta element) 41

I

id attribute
in card element 17
i element 31
i example 31
img element 16, 32, 66
embedded inside of do element 22
img example 34
iname attribute (select element) 56
input element 15, 16, 26, 35
input example 37
intrinsic events
ONCLICK 44
ONENTERBACKWARD 44
ONENTERFORWARD 44
ONTIMER 44
ivalue attribute (select element) 56

L

label attribute
in optgroup element 45
label attribute (do element) 23
less than character (<), specifying in text 7
line breaks
in WML code 4
link element 30, 38
link example 38
list attribute (input element) 56
list attribute (select element) 56
localsrc attribute (img element) 32

M

maxlength attribute (input element) 36
meta element 11, 29, 41
method attribute
in spawn element 62
method attribute (go element) 27
mobile browser device 1
mode attribute (br element) 48
multiple attribute (select element) 55

N

name attribute
  in catch element 20
  in do element 23
  in input element 35
  in meta element 41
  in postfield element 48
  in receive element 52
  in select element 55
  in setvar element 60
  in throw element 68
  in timer element 69
newcontext attribute (card element) 17
newline characters, in WML code 4
non-breaking space character, specifying in text 7
noop element 19, 22, 43, 44

O

onenterbackward attribute
  in card element 17
  in template element 67
onenterforward attribute
  in card element 17
  in template element 67
onevent element 16, 17, 19, 44, 46, 67
  in catchelement 20
onexit attribute
  in spawn element 62
onpick attribute (option element) 46
onthrow attribute
  in catch element 20
ontimer attribute
  in card element 17, 69
  in template element 67
Openwave extension, image element inside option 46
Openwave extension, image inside do element 22, 32
Openwave extensions 5, 20, 25, 30, 38, 52, 54, 58, 61, 68
Openwave extensions DOCTYPE 5
Openwave Mobile Access Gateway platform overview 1
optgroup element 45, 55
optional attribute (do element) 23
option element 44, 45, 46, 55, 57
option example 47
ordered attribute (card element) 17

P

path attribute (access element) 11
p element 16, 48
popup attribute (input element) 56
popup attribute (select element) 56
postfield element 48
prev element 12, 17, 19, 22, 44, 50, 60
prev example 50
prev without content example 51
property attribute (meta element) 41

Q

quote character ("), specifying in text 7

R

radio attribute (input element) 56
radio attribute (select element) 56
receive element 20
refresh element 12, 19, 22, 44, 53, 60
refresh example 53
rel attribute
  in link element 38
reset element 54
reset example 54

S

scheme attribute (meta element) 41
select element 15, 16, 26, 46, 55
select example 57
send element 58
sendelement, embedding in exit element 25
send example 59
sendreferer attribute
  in link element 38
  in spawn element 62
sendreferer attribute (go element) 27
setvar element 27, 50, 53, 60
size attribute (input element) 36
small element 60
soft (discretionary) hyphen, specifying in text 7, 77
source attribute (img element) 30, 32
spaces, in WML code 4
spawn element 12, 20, 22, 44, 60, 61
spawn example 62
special characters
  specifying in text 7
  specifying in URLs 7
spin attribute (select element) 56

strong element 64

T

tabindex attribute (select element) 56
table element 65
table example 65
tabs in WML code 4
task elements 69
    exit task 12, 22, 44
    go task 12, 17, 19, 22, 27, 44
    noop task 19, 22, 43, 44
    prev task 12, 17, 19, 22, 44, 50
    receive element 52
    refresh task 12, 19, 22, 44, 53
    spawnh task 12, 22, 44
    spawn task 12, 22, 44
    throw task 12, 22, 44
td element 66
template element 16, 23, 44, 67
text, using special characters 7
throw element 12, 22, 44, 68
timer element 16, 17, 44, 69
timer example 69
time to live (TTL), setting 41
title attribute
    in a element 10
    in anchor element 12
    in card element 17, 42
    in fieldset element 26
    in input element 35
    in option element 46
    in select element 55
    in table element 65
tr element 66
type attribute
    in do element 23, 44
    in input element 35

U

u element 70

V

value attribute
    in input element 35
    in option element 46, 55
    in select element 56
    in send element 58
    in setvar element 60
    in timer element 69
value attribute (postfield element) 48
variables
    naming 4
    referencing 6
vspace attribute (img element) 32

W

white space, in WML code 4
width attribute (img element) 32
WML character set 4
wml element 16, 70
wml example 70
WML language 1
WML services 2

X

xml:lang attribute (wml element) 70